Statikus forráskód elemzés a mintafelismerésben, a performancia optimalizálásban és a szoftver karbantarthatóságban

Bán Dénes

Szoftverfejlesztés Tanszék
Szegedi Tudományegyetem

Szeged, 2017

Témavezető:

Dr. Ferenc Rudolf

Ph.D. értekezés tézisei

Szegedi Tudományegyetem
Informatika Doktori Iskola

Bevezetés

Szoftverek uralják a világot.

Ez az állítás lehet, hogy már évtizedekkel ezelőtt is fedte volna a valóságot, de napjainkban mindenképp. Amikor az USA lakosságának körében a *bármilyen* beépített rendszert használók aránya – beleértve a telefonokat, fényképezőgépeket, órákat, szórakoztató elektronikát, stb. – a 2011-ben becsült 65%-ról 95%-ra nőtt 2017-re pusztán a telefonoknak köszönhetően. Amikor egyáltalán már *beszélhetünk* „okos városok" építéséről. Amikor – a Cisco adatai szerint – a hálózati eszközök már évek óta legalább másfélszer annyian vannak, mint az emberek.

Ezt csak tovább erősíti az a sok beépített rendszer, amit a legtöbb ember figyelembe sem vesz. Háztartási gépek mint a sütők vagy a hűtők, fűtési és klíma berendezések, a járműveink elindítása vagy épp megállítása mind részei egy mindennapi rutinnak. Egy modern élet – a megszokott és jól látható példákon kívül is – tele van láthatatlan, rejtett processzorokkal. És akkor még nem is említettünk olyan kritikus alkalmazásokat, mint a légi irányítás, egészségügyi eszközök, vagy akár az atomerőművek üzemeltetése.

Ezeknek mind szoftverekre van szükségük a működésükhöz, amit valakinek el is kell készítenie. Sem a szoftveripar növekedése, sem ennek a növekedésnek a folyamatos gyorsulása nem kérdéses. Az egyetlen kérdés, hogy egyáltalán tudjuk-e tartani a lépést.

Miután kellően megalapoztuk a szoftverfejlesztés fontosságát, koncentrálhatunk annak sikeréhez szükséges – vitathatóan – két legfontosabb tényezőre: a karbantarthatóságra és a teljesítményre. A szoftver rendszerek élettartamuk nagy részét a karbantartási fázisban töltik, ami átlagosan a teljes költségek 60%-áért is felelhet. Természetesen egy kódbázis karbantartása nem csak hibajavításból áll, hiszen a fejlesztések és a folyamatosan változó követelmények sokkal gyakoribbak. Mindez azt jelenti, hogy egy hatékony karbantartási ciklushoz egy szoftver terméknek – többek között – könnyen elemezhetőnek, módosíthatónak és tesztelhetőnek kell lennie.

Hasonlóan fontos kérdés a szoftverek teljesítménye és energiahatékonysága. Egy alulteljesítő szoftver termék komoly veszteségekhez vezethet például határidő csúszásokkal, túlköltésekkel, csökkentett produktivitással, rossz vevői véleményekkel, vagy akár elszalasztott piaci lehetőségekkel és bevételekkel. Másrészt a számítóközpontok és adattárházak által felhasznált nagyüzemi mennyiségű energiafogyasztás is egyre nagyobb aggodalomra adhat okot egy olyan társadalomban, ami egyre inkább támaszkodik a számítástechnikára.

Sok akadály áll azonban a „tiszta", karbantartható és nagy teljesítményű szoftverek készítésének útjában. A növekedő piac által diktált határidők miatt gyakran lehetetlennek tűnhet robusztus tervezési gyakorlatokat alkalmazni, amikor ezek késleltethetnék a kiadást. Hasonló módon a sietség és az „előre dolgozás" szándékának hiánya azok, amik antimintákhoz és kód duplikációkhoz vezetnek, ezzel – hosszú távon – csökkentve a minőséget. Az elégtelen hozzáférhetőség és eszköztámogatás pedig komolyan hátráltathatja a fejlesztőket abban, hogy kihasználhassák napjaink teljesítmény optimalizációs lehetőségeit, mint például a specializált gyorsító hardvereket (GPGPU, DSP, FPGA) és a hozzájuk kapcsolódó keretrendszereket.

Jelen munkánkkal ezeket a területeket szeretnénk segíteni. Célunk, hogy:

I. **Felhívjuk a figyelmet a karbantartási fázis, illetve az azt segítő vagy hátráltató tényezők fontosságára azzal, hogy objektív összefüggéseket mutatunk be bizonyos forráskód minták és a karbantarthatóság között; illetve**

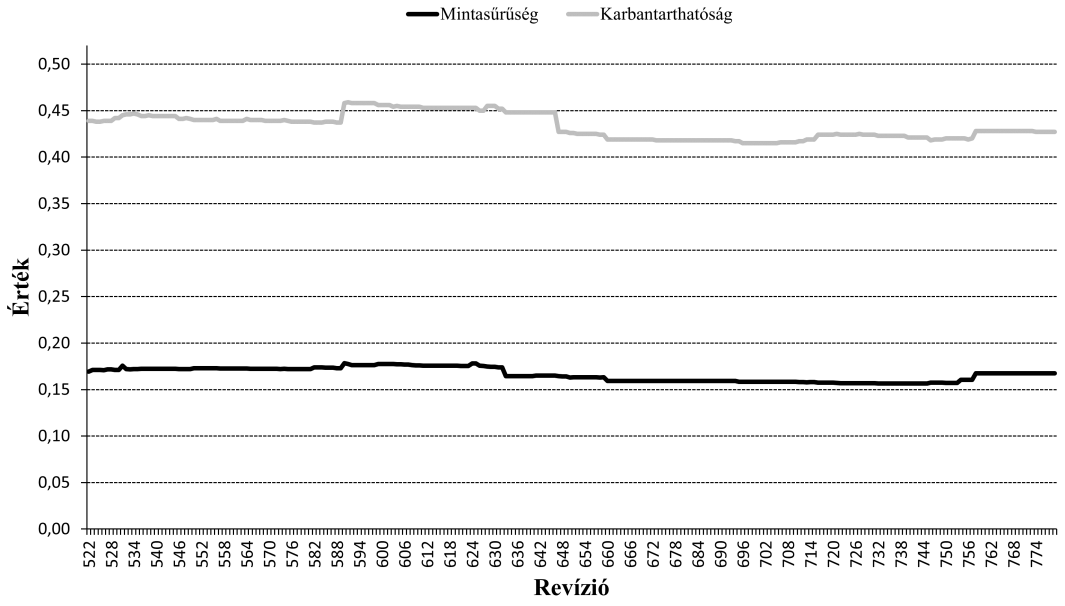
II. **Segítsük a fejlesztőket, hogy könnyebben kihasználhassák a performancia növelésére szolgáló modern gyorsító hardverek nyújtotta lehetőségeket azáltal, hogy bemutatunk egy egyszerűen használható és bővíthető statikus platform választó keretrendszert.**

I. A forráskód minták szoftver karbantarthatóságra kifejtett hatásának empirikus validációja

A tézispont témája a szoftver karbantarthatóság, illetve annak kapcsolatai bizonyos statikus forráskód mintákhoz.

A tervezési minták és a karbantarthatóság kapcsolata

A tervezési minták karbantarthatóságra kifejtett hatásának vizsgálata érdekében a JHotDraw grafikus szoftver több mint 700 revízióját elemeztük [9]. Ezt a rendszert kifejezetten azért választottuk, mert készítői a benne szereplő tervezési mintákat a forráskódban alaposan és következetesen dokumentálták, így az általános felismerő eszközök helyett egy *javadoc* alapú szöveges feldolgozó script-et használhattunk. Ez gyakorlatilag garantálta a kibányászott mintapéldányok precizitását, amit mi egy objektív karbantarthatósági modell használatával egészítettünk ki [1]. Ezután tanulmányoztuk azokat a revíziókat, ahol növekedés történt a rendszerben található tervezési minták számában, és egyértelmű javulást tapasztaltunk a karbantarthatósági értékekben is. Továbbá, a mintasűrűség és a karbantarthatóság átfogó összehasonlítása – 1. ábra – egy 0,89-es Pearson korrelációs együtthatót eredményezett, ami arra utal, hogy a tervezési minták valóban jótékony hatással vannak a karbantarthatóságra.

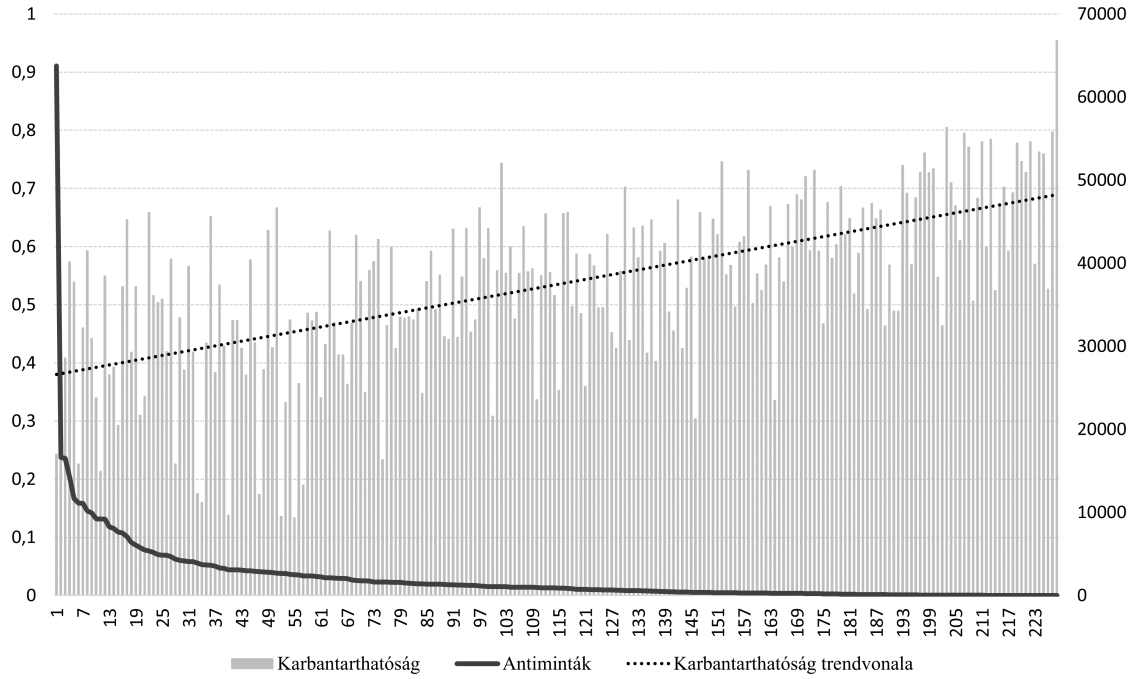


1. ábra. A mintasűrűség és a karbantarthatóság tendenciái

Az antiminták és a karbantarthatóság kapcsolata

Az antiminták vizsgálatára 228 nyílt forráskódú Java rendszert, valamint a Firefox C++ alapú böngésző 45 revízióját választottuk, 2 különálló kísérletre. Mindkét esetben 9, széles körben elterjedt antiminta típust nyertünk ki metrika határszámok és strukturális kapcsolatok alapján – valamint a C++ esetében antiminta sűrűségeket is [7]. Java rendszerekre továbbra is az előző karbantarthatósági modellünket, míg a Firefox kiértékeléséhez egyedi, C++ specifikus minőségi

modellt és a „hagyományos” MI metrika [8] több verzióját használtuk. Mindkét tanulmány az antiminták negatív hatását támasztja alá. A Java rendszerek antiminta tartalmuk szerinti csökkenő sorrendje esetén a karbantarthatósági érték trendvonala egyértelmű javulást mutat, ahogy azt a 2. ábrán is láthatjuk. Az antiminták és a karbantarthatóság közötti átfogó Spearman korreláció itt -0,62 volt.



2. ábra. A karbantarthatóság trendje csökkenő antiminták esetében (Java)

Másrészről a C++ alapú elemzés abszolút és sűrűségi antiminta eredményekhez is vezetett. Ezekből az összegzett antiminta darabszám és sűrűség illetve a végső Karbantarthatóság érték közti korrelációt emelnénk ki, mivel ezek jelképezik legjobban az antiminták és a karbantarthatóság közti átfogó kapcsolatot. Az ide tartozó együtthatók rendre -0,658 és -0,692 voltak Pearson, illetve -0,704 és -0,678 Spearman korreláció esetén. A különböző antiminták és minőségi mutatók közti Spearman korrelációkat az 1. táblázat mutatja.

Egy másik érdekes eredmény, hogy a C++-beli antiminta találatokat karbantarthatósági prediktorokként használva 0,76 és 0,93 közötti pontosságú gépi tanulási modelleket tudtunk építeni.

Az antiminták és a programhibák kapcsolata

Kiegészítésként a Java alapú vizsgálatunk során az antiminták és a programhibák (vagy „bugok”) között is kapcsolatot kerestünk a PROMISE nyílt hiba adatbázis segítségével [12]. A fent említett 228-ból azt a 34 Java rendszert vizsgálva, amikhez hiba információk is tartoztak, statisztikailag szignifikáns 0,55-ös erősségű Spearman korrelációt találtunk az antiminták és a hibák száma között. Ezen felül kimutattuk, hogy az antiminták 67%-os pontossággal tudják előrejelezni a programhibák számát, ami jelentősen jobb az 50%-os alapértéknél, és nem sokkal marad el ötször több statikus forráskód metrika 71,2%-os teljesítményétől sem.

Antiminta	MI	MI_2	MI*	MI_2*	**Elemezhetőség**	**Módosíthatóság**	**Modularitás**	**Újrahasznosíthatóság**	**Tesztelhetőség**	**Karbantarthatóság**
FE	−,985**	−,985**	−,853**	−,809**	−,852**	−,749**	−,161	−,370*	−,806**	−,652**
FE_{DENS}	−,535**	−,535**	−,257	−,258	−,440**	−,532**	−,550**	−,595**	−,561**	−,609**
LC	−,757**	−,757**	−,936**	−,920**	−,755**	−,538**	,224	−,044	−,572**	−,381**
LC_{DENS}	−,542**	−,542**	−,777**	−,854**	−,517**	−,276	,372*	,123	−,307*	−,133
LCC	−,985**	−,985**	−,873**	−,839**	−,871**	−,754**	−,131	−,354*	−,811**	−,651**
LCC_{DENS}	−,482**	−,482**	−,213	−,258	−,439**	−,543**	−,590**	−,655**	−,550**	−,636**
LCD	−,731**	−,731**	−,484**	−,445**	−,509**	−,551**	−,303*	−,365*	−,590**	−,528**
LCD_{DENS}	−,626**	−,626**	−,355*	−,344*	−,373*	−,431**	−,299*	−,318*	−,474**	−,428**
LF	−,991**	−,991**	−,849**	−,800**	−,902**	−,821**	−,242	−,453**	−,866**	−,728**
LF_{DENS}	−,874**	−,874**	−,622**	−,608**	−,824**	−,837**	−,500**	−,671**	−,876**	−,821**
LPL	−,952**	−,952**	−,926**	−,856**	−,851**	−,696**	,019	−,219	−,750**	−,560**
LPL_{DENS}	−,904**	−,904**	−,707**	−,670**	−,715**	−,654**	−,184	−,338*	−,708**	−,580**
RB	−,976**	−,976**	−,819**	−,793**	−,829**	−,734**	−,167	−,375*	−,794**	−,646**
RB_{DENS}	−,911**	−,911**	−,706**	−,728**	−,735**	−,674**	−,219	−,396**	−,730**	−,614**
SHS	−,985**	−,985**	−,820**	−,768**	−,884**	−,827**	−,291	−,487**	−,871**	−,747**
SHS_{DENS}	−,907**	−,907**	−,694**	−,698**	−,787**	−,773**	−,370*	−,538**	−,812**	−,726**
SUM	−,978**	−,978**	−,806**	−,754**	−,847**	−,786**	−,250	−,444**	−,834**	−,704**
SUM_{DENS}	−,895**	−,895**	−,674**	−,641**	−,732**	−,726**	−,332*	−,476**	−,768**	−,678**
TF	−,945**	−,945**	−,746**	−,704**	−,785**	−,746**	−,277	−,445**	−,796**	−,681**
TF_{DENS}	−,843**	−,843**	−,595**	−,543**	−,675**	−,704**	−,378*	−,484**	−,739**	−,665**

1. táblázat. Spearman korrelációk különböző antiminta és karbantarthatósági értékek között (C++)

Módszer	TP arány	FP arány	Precision	Recall	F-Measure
Antiminták	0,658	0,342	0,670	0,658	0,653
Metrikák	0,711	0,289	0,712	0,711	0,711
Mindkettő	0,712	0,288	0,712	0,712	0,712

2. táblázat. A hiba előrejelző kísérletek eredményei

A szerző hozzájárulása

A tervezési mintákkal kapcsolatos kutatásban a szerző főként a mintafelismerő eszköz implementációjához, a forráskód metrikák kiszámításához, a mintapéldányok számának változásával járó revíziók kézi ellenőrzéséhez és a kapcsolódó irodalom feldolgozásához járult hozzá. Ezzel szemben mindkét antimintákkal kapcsolatos tanulmány teljes egészében a szerző munkája, beleértve a rendszerek előkészítését és elemzését, a statikus forráskód metrikák implementációját és kinyerését, a karbantarthatósági értékek kiszámítását, – a C++ specifikus minőségi modell készítésével együtt – az antiminták értelmezését, implementálását és beazonosítását, a programhiba információk feldolgozását, valamint az empirikus kísérletek megtervezését és lebonyolítását is. A tézispont a következő publikációkra épül:

- Péter Hegedűs, **Dénes Bán**, Rudolf Ferenc, and Tibor Gyimóthy. Myth or Reality? Analyzing the Effect of Design Patterns on Software Maintainability. In *Advanced Software Engineering & Its Applications (ASEA 2012), Jeju Island, Korea, November 28 – December 2*, pages 138–145, CCIS, Volume 340. Springer Berlin Heidelberg, 2012.

- **Dénes Bán** and Rudolf Ferenc. Recognizing Antipatterns and Analyzing their Effects on Software Maintainability. In *14th International Conference on Computational Science and Its Applications (ICCSA 2014), Guimarães, Portugal, June 30 – July 3*, pages 337–352, LNCS, Volume 8583. Springer International Publishing, 2014.

- **Dénes Bán**. The Connection of Antipatterns and Maintainability in Firefox. In *10th Jubilee Conference of PhD Students in Computer Science (CSCS 2016), Szeged, Hungary, June 27 – 29*, 2016.

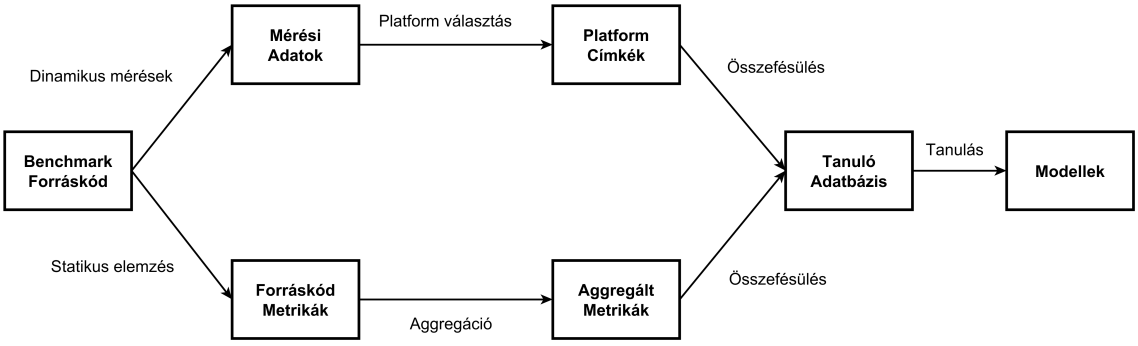
- **Dénes Bán**. The Connection of Antipatterns and Maintainability in Firefox. Közlésre elfogadva az Acta Cybernetica 2016-os különkiadásában (a fenti CSCS 2016 publikáció kibővített változata). 20 oldal.

II. Performancia optimalizációt elősegítő, statikus forráskód metrikákon alapuló hardver platform választó keretrendszer

A tézispont témája a szoftver performancia és egy automatikus hardver platform választó módszer.

Kvalitatív modellek

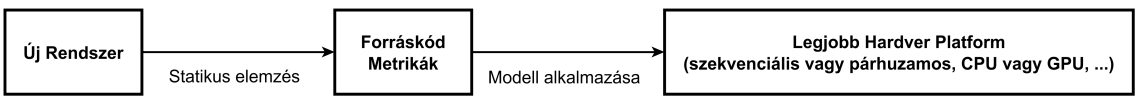
A itt bemutatott kutatásunk fő célja egy általánosítható módszertan kidolgozása volt olyan modellek építéséhez, amik képesek tisztán statikus információk alapján megbecsülni, hogy egy adott programot várhatóan melyik hardver platformon lehet optimálisan végrehajtani – mind a futásidő, mind pedig az energia fogyasztás szemszögéből. Ennek alapjául számos „benchmark" programot gyűjtöttük, bennük olyan algoritmusokkal, amik minden lehetséges célplatformhoz rendelkeztek implementációval. A modelljeink betanításához szükség volt ilyen algoritmusokra, hiszen a performanciabeli eltérésekre úgy világíthatunk rá, ha a különböző verzióikat a kapcsolódó platformjaikon futtatjuk. Ezután számos (alacsony szintű) forráskód metrikát nyertünk ki ezekből az algoritmusokból, amik jól megragadják a jellemzőiket és modelljeink prediktorai lehetnek. Emellett egy olyan általános megoldást is kifejlesztettünk, ami képes pontos, platformfüggetlen idő és energiamérésekre [11]. Végül különböző gépi tanulási módszereket használva megépíthettük a célként kitűzött modelleket. Egy rövid empirikus validáció igazolta a modellek elméleti hasznosságát, – amelyek néhol 100%-os pontosságot is elértek – de a tanulmány igazi eredménye a módszertan, – 3. ábra – amivel megépítettük őket, és ami nagyobb szabású kísérletekre is lehetőséget adhat. Az épített modellek új (a modell számára ismeretlen) rendszereken való alkalmazását a 4. ábra mutatja be.



3. ábra. A modellépítő folyamat fő lépései

Kvantitatív modellek

Az előző eredményeinkre építve úgy bővítettük a módszerünket, hogy már kvantitatív modellek készítésére is képes legyen, amik nem csak a legjobb platformot becsülik meg, hanem az ott



4. ábra. Korábban épített modellek használata új rendszereken

várható teljesítmény növekedés arányát is. Emellett jelentősen megnöveltük a kinyert forráskód metrikáink számát, pontosabb metrika kinyerési stratégiát dolgoztunk ki, – elválasztva a benchmarkok algoritmusainak lényegi részeit, avagy „kerneleket" – új benchmark-okkal bővítettük az elemzett rendszereinket, és lehetséges platformként bevezettük az FPGA-kat is. Az elért előrejelzési pontosságokat a teljes rendszerekre, illetve csak a kernelekre vonatkozóan rendre a 3. és a 4. táblázat mutatja be. A hátrom fejléc réteg a dinamikus mérések aggregálási módszerét, a célplatformot és a mért aspektust (**I**dő, **T**eljesítmény, **E**nergia) képviseli. Mivel a javulási arányok folytonosak, így közelítésükhöz regressziós algoritmusokat (felső hat sor), valamint diszkretizáló előfeldolgozás után osztályozó algoritmusokat is használhattunk – 5 kategóriával a középső öt sorban, illetve 3 kategóriával az alsó ötben. Habár a regressziók ritkán vezettek biztató eredményekhez, az osztályozások 94%-a legalább 5%-kal (és időnként akár 49%-kal) pontosabb tudott lenni a véletlenszerű választásnál és az alapkonfigurációnál is.

A forráskód párhuzamosítás és a karbantarthatóság kapcsolata

A rendelkezésre álló benchmark forráskódok lehetővé tették azt is, hogy megvizsgáljuk a CPU alapú eredeti (szekvenciális) és a gyorsító hardver specifikus (párhuzamos) algoritmus verziók közti karbantarthatóság különbségeket. Az egyetlen további előfeltétel az volt, hogy a párhuzamos verziók forráskódjából is kinyerjük a korábbi metrikákat, hiszen a minőségi modellt felhasználhattuk egy korábbi tanulmányból [13]. Az összehasonlítás eredményei azt mutatták, hogy a párhuzamosított implementációk karbantarthatósága jelentősen alacsonyabb, mint a szekvenciális párjaiké (lásd az 5. táblázatban). Ez azonban nem volt olyan egyértelműen kimutatható az algoritmusok lényegi részében (kernelekben), – lásd a 6. táblázat – ami arra utal, hogy a minőségromlást főként a felhasznált, gyorsító hardver specifikus keretrendszerek által bevezetett extra infrastruktúra (boilerplate) okozza.

A szerző hozzájárulása

A benchmark-ok gyűjtése és előkészítése – mind statikus, mind dinamikus elemzésre – a szerző vezetésével történt. Ő implementálta, nyerte ki, és aggregálta az eredeti és a kibővített forráskód metrikákat, és ő állította össze a gépi tanuláshoz használatos táblázatokat. Ő formalizálta és végezte el az empirikus kísérleteket, és elemezte az eredményeiket. Az algoritmus verziók karbantarthatóságának összehasonlítása és kiértékelése szintén a szerző munkája. A tézispont a következő publikációkra épül:

- **Dénes Bán**, Rudolf Ferenc, István Siket, and Ákos Kiss. Prediction Models for Performance, Power, and Energy Efficiency of Software Executed on Heterogeneous Hardware. In *13th IEEE International Symposium on Parallel and Distributed Processing with Applications (IEEE ISPA-15), Helsinki, Finland, August 20 – 22*, pages 178–183, IEEE Trustcom/-BigDataSE/ISPA, Volume 3. IEEE Computer Society Press, 2015.

- **Dénes Bán**, Rudolf Ferenc, István Siket, Ákos Kiss, and Tibor Gyimóthy. Prediction Models for Performance, Power, and Energy Efficiency of Software Executed on Heterogeneous Hardware. Elbírálás alatt a Journal of Supercomputing-nál, Springer Publishing (a fenti IEEE ISPA-15 publikáció kibővített változata). 24 oldal.

	Egyedül									CPU-val									Mind								
	CPU			GPU			FPGA			CPU			GPU			FPGA			CPU			GPU			FPGA		
Algoritmus	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**
ZeroR	0,65	0,45	0,65	0,39	0,45	0,42	0,65	0,39	0,65	0,65	0,45	0,65	0,39	0,37	0,40	0,65	0,35	0,65	0,65	0,44	0,65	0,39	0,37	0,40	0,65	0,38	0,65
LinReg	0,65	0,45	0,65	0,39	0,19	0,42	0,65	0,39	0,65	0,65	0,45	0,65	0,39	0,37	0,40	0,65	0,35	0,65	0,65	0,44	0,65	0,39	0,30	0,40	0,65	0,38	0,65
Mult.Perc.	0,01	0,37	0,01	0,12	0,66	0,44	0,09	0,07	0,10	0,01	0,37	0,01	0,12	0,42	0,15	0,09	0,34	0,12	0,01	0,45	0,03	0,12	0,62	0,15	0,09	0,15	0,10
REPTree	0,10	0,45	0,12	0,63	0,26	0,83	0,23	0,36	0,25	0,10	0,45	0,12	0,63	0,39	0,70	0,23	0,05	0,02	0,10	0,39	0,12	0,63	0,26	0,70	0,23	0,08	0,25
M5P	0,30	0,13	0,32	0,65	0,50	0,79	0,47	0,10	0,48	0,30	0,13	0,32	0,65	0,17	0,72	0,47	0,13	0,48	0,30	0,12	0,32	0,65	0,28	0,71	0,47	0,01	0,48
SMOreg	0,06	0,25	0,08	0,15	0,70	0,10	0,04	0,37	0,04	0,06	0,25	0,08	0,15	0,65	0,15	0,04	0,20	0,04	0,06	0,30	0,08	0,15	0,64	0,16	0,04	0,20	0,04
ZeroR	16,28	16,28	16,28	11,11	11,11	11,11	0,00	0,00	0,00	16,28	16,28	16,28	11,11	11,11	11,11	0,00	0,00	0,00	16,28	16,28	16,28	11,11	11,11	11,11	0,00	0,00	0,00
J48	37,21	27,91	58,14	53,33	60,00	53,33	34,48	27,59	37,93	37,21	27,91	58,14	53,33	51,11	60,00	34,48	41,38	48,28	37,21	25,58	58,14	53,33	28,89	60,00	34,48	34,48	41,38
NaiveBayes	25,58	30,23	20,93	22,22	28,89	22,22	17,24	37,93	24,14	25,58	30,23	20,93	22,22	17,78	17,78	17,24	31,03	20,69	25,58	30,23	20,93	22,22	20,00	17,78	17,24	24,14	13,79
Logistic	27,91	23,26	30,23	51,11	31,11	40,00	27,59	37,93	31,03	27,91	23,26	30,23	51,11	33,33	46,67	27,59	31,03	24,14	27,91	27,91	30,23	51,11	33,33	46,67	27,59	24,14	24,14
SMO	39,53	27,91	27,91	40,00	28,89	42,22	27,59	41,38	17,24	39,53	27,91	27,91	40,00	26,67	44,44	27,59	17,24	3,45	39,53	23,26	27,91	40,00	17,78	44,44	27,59	13,79	10,34
ZeroR	23,26	23,26	23,26	22,22	22,22	22,22	34,48	34,48	34,48	23,26	23,26	23,26	22,22	22,22	22,22	34,48	34,48	34,48	23,26	23,26	23,26	22,22	22,22	22,22	34,48	34,48	34,48
J48	65,12	41,86	65,12	71,11	57,78	60,00	55,17	37,93	55,17	65,12	41,86	65,12	71,11	57,78	71,11	55,17	44,83	55,17	65,12	46,51	65,12	71,11	44,44	71,11	55,17	34,48	55,17
NaiveBayes	32,56	37,21	32,56	33,33	40,00	40,00	58,62	41,38	58,62	32,56	37,21	32,56	33,33	44,44	33,33	58,62	41,38	58,62	32,56	37,21	32,56	33,33	46,67	33,33	58,62	37,93	58,62
Logistic	34,88	51,16	34,88	66,67	46,67	60,00	62,07	48,28	62,07	34,88	51,16	34,88	66,67	60,00	66,67	62,07	41,38	62,07	34,88	53,49	34,88	66,67	57,78	66,67	62,07	41,38	62,07
SMO	39,53	46,51	39,53	53,33	60,00	42,22	55,17	55,17	55,17	39,53	46,51	39,53	53,33	46,67	53,33	55,17	55,17	55,17	39,53	44,19	39,53	53,33	53,33	53,33	55,17	41,38	55,17

3. táblázat. Teljes programokra vonatkozó előrejelzési pontosságok

	Egyedül									CPU-val									Mind								
	CPU			GPU			FPGA			CPU			GPU			FPGA			CPU			GPU			FPGA		
Algoritmus	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**	**I**	**T**	**E**
ZeroR	0,48	0,46	0,48	0,36	0,46	0,37	0,65	0,41	0,65	0,48	0,46	0,48	0,36	0,42	0,36	0,65	0,41	0,65	0,48	0,47	0,48	0,36	0,44	0,35	0,65	0,41	0,65
LinReg	0,48	0,46	0,48	0,36	0,32	0,37	0,65	0,41	0,65	0,48	0,46	0,48	0,36	0,42	0,36	0,65	0,41	0,65	0,48	0,47	0,48	0,36	0,23	0,35	0,65	0,41	0,65
Mult.Perc.	0,01	0,13	0,02	0,63	0,63	0,56	0,07	0,16	0,16	0,01	0,13	0,02	0,63	0,46	0,67	0,07	0,39	0,13	0,01	0,14	0,02	0,63	0,50	0,61	0,07	0,48	0,06
REPTree	0,08	0,10	0,08	0,74	0,27	0,70	0,19	0,18	0,21	0,08	0,10	0,08	0,74	0,00	0,77	0,19	0,07	0,13	0,08	0,47	0,08	0,74	0,21	0,74	0,19	0,19	0,21
M5P	0,01	0,04	0,01	0,67	0,05	0,67	0,43	0,16	0,44	0,01	0,04	0,01	0,67	0,02	0,64	0,43	0,16	0,44	0,01	0,04	0,01	0,67	0,11	0,67	0,43	0,16	0,44
SMOreg	0,04	0,07	0,05	0,00	0,53	0,00	0,07	0,12	0,07	0,04	0,07	0,05	0,00	0,62	0,01	0,07	0,04	0,06	0,04	0,12	0,05	0,00	0,63	0,01	0,07	0,08	0,06
ZeroR	16,28	16,28	16,28	11,11	11,11	11,11	0,00	0,00	0,00	16,28	16,28	16,28	11,11	11,11	11,11	0,00	0,00	0,00	16,28	16,28	16,28	11,11	11,11	11,11	0,00	0,00	0,00
J48	32,56	18,60	44,19	48,89	35,56	46,67	68,97	20,69	68,97	32,56	18,60	44,19	48,89	22,22	48,89	68,97	17,24	68,97	32,56	34,88	37,21	48,89	48,89	48,89	68,97	48,28	68,97
NaiveBayes	18,60	20,93	13,95	26,67	31,11	22,22	34,48	24,14	34,48	18,60	20,93	13,95	26,67	20,00	33,33	34,48	17,24	34,48	18,60	13,95	6,98	26,67	28,89	33,33	34,48	27,59	34,48
Logistic	41,86	25,58	44,19	33,33	37,78	40,00	41,38	27,59	41,38	41,86	25,58	44,19	33,33	33,33	35,56	41,38	24,14	41,38	41,86	25,58	25,58	33,33	31,11	35,56	41,38	27,59	41,38
SMO	32,56	20,93	20,93	33,33	28,89	40,00	34,48	20,69	34,48	32,56	20,93	20,93	33,33	20,00	42,22	34,48	17,24	34,48	32,56	23,26	20,93	33,33	28,89	42,22	34,48	10,34	34,48
ZeroR	30,23	23,26	30,23	22,22	22,22	22,22	34,48	34,48	34,48	30,23	23,26	30,23	22,22	22,22	22,22	34,48	34,48	34,48	30,23	23,26	30,23	22,22	22,22	22,22	34,48	34,48	34,48
J48	55,81	32,56	55,81	53,33	62,22	53,33	55,17	48,28	55,17	55,81	32,56	55,81	53,33	46,67	48,89	55,17	41,38	55,17	55,81	32,56	55,81	53,33	35,56	48,89	55,17	37,93	55,17
NaiveBayes	37,21	39,53	37,21	44,44	44,44	57,78	34,48	31,03	34,48	37,21	39,53	37,21	44,44	40,00	48,89	34,48	17,24	34,48	37,21	39,53	37,21	44,44	42,22	48,89	34,48	17,24	34,48
Logistic	65,12	37,21	65,12	48,89	55,56	53,33	68,97	37,93	68,97	65,12	37,21	65,12	48,89	57,78	53,33	68,97	27,59	68,97	65,12	37,21	65,12	48,89	53,33	53,33	68,97	34,48	68,97
SMO	53,49	46,51	53,49	48,89	55,56	51,11	44,83	41,38	44,83	53,49	46,51	53,49	48,89	40,00	55,56	44,83	17,24	44,83	53,49	46,51	53,49	48,89	44,44	55,56	44,83	27,59	44,83

4. táblázat. Kernelekre vonatkozó előrejelzési pontosságok

Benchmark	Elemezhetőség	Módosíthatóság	Modularitás	Újrahasznosíthatóság	Tesztelhetőség	Karbantarthatóság
mri-q	−0,388	−0,405	−0,448	−0,432	−0,360	−0,407
spmv	−0,667	−0,685	−0,653	−0,676	−0,658	−0,668
stencil	−0,225	−0,237	−0,428	−0,325	−0,199	−0,283
atax	−0,338	−0,354	−0,472	−0,406	−0,298	−0,375
bicg	−0,342	−0,358	−0,471	−0,412	−0,308	−0,379
conv2d	−0,332	−0,346	−0,469	−0,405	−0,296	−0,370
doitgen	−0,372	−0,388	−0,582	−0,476	−0,324	−0,429
gemm	−0,269	−0,283	−0,435	−0,352	−0,237	−0,315
gemver	−0,325	−0,343	−0,494	−0,417	−0,294	−0,375
gesummv	−0,290	−0,304	−0,384	−0,343	−0,262	−0,317
jacobi2d	−0,420	−0,433	−0,560	−0,491	−0,373	−0,456
mvt	−0,339	−0,353	−0,444	−0,396	−0,304	−0,368
bfs	−0,352	−0,367	−0,497	−0,431	−0,319	−0,393
hotspot	−0,226	−0,235	−0,371	−0,308	−0,167	−0,261
lavaMD	−0,271	−0,276	−0,352	−0,315	−0,244	−0,292
nn	−0,429	−0,434	−0,560	−0,485	−0,364	−0,456

5. táblázat. Rendszerszintű karbantarthatóság változások

Benchmark	Elemezhetőség	Módosíthatóság	Modularitás	Újrahasznosíthatóság	Tesztelhetőség	Karbantarthatóság
mri-q	−0,234	−0,240	−0,395	−0,321	−0,225	−0,282
spmv	0,139	0,135	−0,308	−0,069	0,188	0,019
stencil	0,145	0,144	−0,617	−0,205	0,220	−0,059
atax	−0,109	−0,136	−0,435	−0,283	−0,087	−0,208
bicg	−0,200	−0,222	−0,449	−0,329	−0,162	−0,272
conv2d	−0,065	−0,075	−0,431	−0,228	−0,002	−0,161
doitgen	0,147	0,131	−0,653	−0,228	0,226	−0,072
gemm	0,120	0,110	−0,391	−0,123	0,175	−0,019
gemver	−0,161	−0,187	−0,708	−0,429	−0,119	−0,319
gesummv	−0,041	−0,055	−0,341	−0,199	−0,033	−0,131
jacobi2d	−0,035	−0,057	−0,691	−0,347	0,019	−0,220
mvt	−0,148	−0,174	−0,443	−0,302	−0,115	−0,236
bfs	0,067	0,063	−0,408	−0,150	0,095	−0,064
hotspot	−0,035	−0,041	−0,774	−0,390	0,043	−0,235
lavaMD	−0,158	−0,165	−0,434	−0,303	−0,150	−0,239
nn	0,015	0,017	0,007	0,006	0,013	0,012

6. táblázat. Kernel szintű karbantarthatóság változások

Összefoglalás

A disszertáció eredményei két fő tézispontba foglalhatók össze.

Az első tézispont fő eredményei maguk az I. szekcióban említett empirikus tanulmányok, amik a forráskód minták és a karbantarthatóság kapcsolatára vonatkozó intuitív elvárásainkat objektív, kézzel fogható adatokkal támasztják alá. Tudomásunk szerint ezeket az eredményeket elsők között sikerült ilyen nagy mennyiségű, nagy méretű és változatos rendszereken, illetve minden szubjektív tényező – például kérdőívek, időkövetés vagy interjúk – nélkül elérnünk.

A második tézispont fő eredményei (a) az empirikus bizonyíték, hogy a statikus forráskód metrikák hasznosak a teljesítmény-javulás előrejelzésében, és (b) egy általános módszertan kvalitatív és kvantitatív hardver platform választó modellek építéséhez. Egy fontos különbség az általunk alkalmazott stratégia és más elérhető megoldások között, hogy a mi modelljeink – megépítésük után – csak statikus információkra hagyatkoznak. Továbbá a pontosságuk leginkább a tanításhoz használt benchmark-ok számának függvénye. Ezek a tulajdonságok teszik a módszerünket egyszerűen továbbfejleszthetővé, a modelljeit pedig egyszerűen alkalmazhatóvá.

A tézispontokat és a kapcsolódó publikációkat a 7. táblázat összegzi.

№	[10]	[4]	[2]	[3]	[5]	[6]
I.	♦	♦	♦	♦		
II.					♦	♦

7. táblázat. A tézispontokhoz kapcsolódó publikációk

Köszönetnyilvánítás

Első sorban szeretném megköszönni témavezetőmnek, Dr. Ferenc Rudolfnak a sok értékes tanácsot és útmutatást. Többször megmutatta a közös kutatásaink alatt, hogy egy rossz eredmény vagy egy sikertelen kísérlet nem egy projekt végét jelenti, csak egy pontot, ahol stratégiát váltunk és folytatjuk tovább. Szintén köszönet illeti Dr. Hegedűs Pétert, amiért a kezdeti lépéseknél „fogta a kezem”, illetve Dr. Siket Istvánt az empirikus eloszlásfüggvényekkel kapcsolatos ötleteiért és amiért mindig rendelkezésre állt, amikor segítségre volt szükségem. Köszönöm még Dr. Gyimóthy Tibornak, a Szoftverfejlesztés Tanszék vezetőjének, amiért lehetőséget biztosított a kutatásaimhoz. Külön köszönet Ladányi Gergelynek a minőségi modellekkel és adatelemzéssel kapcsolatos javaslataiért, Sipka Róbertnek és Molnár Péternek a dinamikus méréseknél végzett fáradhatatlan munkájukért, illetve David Curley-nek a nyelvtani és stilisztikai megjegyzéseiért. Végül köszönet további társszerzőimnek, név szerint Dr. Kiss Ákosnak és Gyimesi Gábornak, a közös eredményekhez való hozzájárulásaikért.

Bán Dénes, 2017

Hivatkozások

[1] Tibor Bakota, Péter Hegedűs, Péter Körtvélyesi, Rudolf Ferenc, and Tibor Gyimóthy. A Probabilistic Software Quality Model. In *Proceedings of the 27th IEEE International Conference on Software Maintenance*, ICSM 2011, pages 368–377, Williamsburg, VA, USA, 2011. IEEE Computer Society.

[2] Dénes Bán. The connection of antipatterns and maintainability in firefox. In *10^{th} Jubilee Conference of PhD Students in Computer Science (CSCS 2016), Szeged, Hungary, June 27 – 29*, 2016.

[3] Dénes Bán. The connection of antipatterns and maintainability in firefox. Accepted for publication in the 2016 Special Issue of Acta Cybernetica (extended version of [2]). 20 pages.

[4] Dénes Bán and Rudolf Ferenc. Recognizing antipatterns and analyzing their effects on software maintainability. In *14^{th} International Conference on Computational Science and Its Applications (ICCSA 2014), Guimarães, Portugal, June 30 – July 3*, pages 337–352. Springer International Publishing, 2014.

[5] Dénes Bán, Rudolf Ferenc, István Siket, and Ákos Kiss. Prediction models for performance, power, and energy efficiency of software executed on heterogeneous hardware. In *13^{th} IEEE International Symposium on Parallel and Distributed Processing with Applications (IEEE ISPA-15), Helsinki, Finland, August 20 – 22*, volume 3, pages 178–183, 2015.

[6] Dénes Bán, Rudolf Ferenc, István Siket, Ákos Kiss, and Tibor Gyimóthy. Prediction models for performance, power, and energy efficiency of software executed on heterogeneous hardware. Submitted to the Journal of Supercomputing (extended version of [5]). 24 pages.

[7] William J. Brown, Raphael C. Malveau, Hays W. McCormick, III, and Thomas J. Mowbray. *AntiPatterns: Refactoring Software, Architectures, and Projects in Crisis*. John Wiley & Sons, Inc., New York, NY, USA, 1998.

[8] D. Coleman, D. Ash, B. Lowther, and P. Oman. Using metrics to evaluate software system maintainability. *Computer*, 27(8):44–49, Aug 1994.

[9] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Pub Co, 1995.

[10] Péter Hegedűs, Dénes Bán, Rudolf Ferenc, and Tibor Gyimóthy. Myth or reality? analyzing the effect of design patterns on software maintainability. In *Advanced Software Engineering & Its Applications (ASEA 2012), Jeju Island, Korea, November 28 – December 2*, pages 138–145. Springer Berlin Heidelberg, 2012.

[11] Ákos Kiss, Péter Molnár, and Róbert Sipka. Rmeasure performance and energy monitoring library. `https://github.com/sed-szeged/RMeasure`, 2017.

[12] Tim Menzies, Bora Caglayan, Zhimin He, Ekrem Kocaguneli, Joe Krall, Fayola Peters, and Burak Turhan. The promise repository of empirical software engineering data, June 2012.

[13] Rudolf Ferenc et al. *REPARA deliverable D7.4: Maintainability models of heterogeneous programming models*. 2015.